RSTS-11 SIG DECUS

FEB 1977 RSTS-11 SPECIAL INTEREST GROUP NEWSLETTER VOL 4 NO 1

FROM THE EDITOR

TAKE THREE HUNDRED RSTS USERS AND PUT THEM TOGETHER FOR FOUR DAYS AT DECUS IN LAS VEGAS AND YOU HAVE A LOT OF INFORMATION SHARED. THE SIG HAS COME A LONG WAY IN FOUR YEARS AND IT IS GREAT TO SEE. WITH OUR LARGE SIZE IT HAS BECOME APPARENT THAT TO HAVE USEFUL MEETINGS AT DECUS WE MUST HAVE SMALLER SESSIONS ON SPECIALIZED TOPICS. WE NEED TO PLAN NOW FOR SPRING-DECUS, AND YOU NEED TO SUBMIT A PAPER OR WORKSHOP FOR YOUR PET AREA OF INTEREST. IF YOU DON'T DO IT, IT WON'T GET DONE. THERE ARE AREAS THAT YOU HAVE KNOWLEDGE IN LIKE SYSTEM PERFORMANCE, FILE HANDLING, WRITING RUN-TIME SYSTEMS OR RUNNING COBOL UNDER RSTS THAT USERS NEED TO HEAR AND TALK ABOUT. THE FIRST PART OF MARCH IS THE PAPER DEADLINE, SO GET THEM IN.

THE SIG DECIDED SEVERAL THINGS DURING ITS GENERAL SESSION. FIRST IT WAS VOTED THAT SCHOOLS PRODUCING RSTS TRAINED STUDENTS WOULD BE THE ONLY USERS ALLOWED TO ADVERTISE POSITIONS OR PEOPLE AVAILABLE FOR POSITIONS IN THE NEWSLETTER. ALSO THE SIG VOTED TO PETITION DEC TO PROVIDE A RSTS SOFTWARE HOTLINE TO MAYNARD. SO FAR WE HAVE NOT GOTTEN A FAVORABLE ANSWER FROM ANYONE AT DEC. I HOPE YOU WILL WRITE THE DEC SOFTWARE MAINTENANCE GROUP IF YOU FEEL AS I DO THAT A HOTLINE IS NECESSARY.

TODAY IS JANUARY 5, 1977 AS A POINT OF REFERENCE. UNFORTUNATELY, BY VIRTUE OF A CATCH 22 IN DECUS WITH THIS ISSUE THE NEWSLETTER IS BACK TO THIRD CLASS MAIL. THE FIRST CLASS ISSUES WERE COSTING DECUS OVER $300 MORE PER ISSUE, SO WE HAVE LOST OUR "CLASS". DECUS IS TRYING TO FIND A WAY TO IMPROVE IT, BUT FRANKLY I'M NOT HOPEFUL.

THE DEADLINE FOR THE NEXT NEWSLETTER IS MARCH 10 - SEND ME SOMETHING.

SEE THE NOTICE BELOW ABOUT THE MEETING IN CHICAGO. IT SHOULD BE REALLY INFORMATIVE SO I HOPE YOU CAN COME. WE EXPECT ABOUT 100 USERS.

BOB BRANTON
R. LABAR & ASSOC.
2750 NORTHAVEN, SUITE 102
DALLAS, TEXAS 75229 (214) 243-3451

****

A RSTS SIG MEETING ON V6B WILL BE HELD AT THE HYATT REGENCY O'HARE IN CHICAGO FROM FEBRUARY 20 THROUGH 22. SEVERAL RSTS DEVELOPMENT PEOPLE WILL BE THERE FROM MAYNARD. FOR MORE INFORMATION WRITE OR CALL:

STUE KATZ
COMMERCIAL COMPUTER SERVICES
401 FULLERTON PARKWAY
CHICAGO, ILLINOIS 60614
(312) 929-7400

****

Here is a list of the hard working people who are chairing RSTS Subgroups:

A. COMMERCIAL RSTS USERS
Stu Katz
Commercial Computer Services
401 Fullerton Parkway
Chicago, Ill. 60614
(312) 929-7400

EDUCATIONAL RSTS USERS
Howard Kipple
New Mexico Junior College
Hobbs, New Mexico 88240
(505) 392-6526

CUSP EDITOR
Jon Spencer
PASCAL Computing
432 S. Barrington Ave.
Los Angeles, Ca. 90049

JSTATS, DSTATS DOCUMENTATION
Mike Mayfield
Digital Equipment Corporation
871 South Nash St.
El Segundo, Ca. 90245
(213) 640-1830

EDITOR OF COMMERCIAL SECTION
Murry Stinson
Scientific Placement
2700 South Post Oak Road
Suite 2430
Houston, Tx 77056
(713) 621-9811

LARGE RSTS SYSTEMS GROUP
Joan Salkin
Crown Zellerbach
1 Bush Street
San Francisco, California 94119
(415) 823-5504

TECHNICAL EDITOR
Carl Marbach
Acme-Mardesty Company
910 Fox Pavilion
Jenkintown, Pa. 19046
(215) 886-0200

B. WORKING GROUPS:

RSTS USER PROFILES
Bill Tabor
Florida Computer, Inc.
99 NW 183rd St.
N. Miama, Florida 33169
(305) 652-1710

RSTS DECUS LIBRARY PROGRAM
Ardoth Wilson
Central State University
100 N. University
Edmond, Ok. 73034
(405) 341-2980 Ext. 321

PROGRAMMING STANDARDS DEVELOPMENT
Roy Swatzell, Jr.
V.A. Hospital
Room 2 R 12
Birmingham, Al 35233
(205) 933-8103 Ext. 239

RSTS USER WI M LIST DEVELOPMENT
Bill Sconce
University of Louisville HSCC
P.O. Box 1055
Louisville, Ky 40201
(502) 588-6911

At Fall DECUS '76 the Library Committee voted to recommend the purchase of a PDP-12 for library use. This will be the answer for many of the media problems for 8's, 12's 15's, 1Ø's and 2Ø's. PDP-11 users will also benefit from it for "like-media" transfer (for example, floppy to floppy or dectape to dectape). Seven and nine track magtape units will be on the system. Any input or suggestions on media transfer from unlike media would be appreciated.

Liaison Support - someone from the different product lines within DEC to interface with the DECUS librarian about 10 hours a month - is being requested. The RSTS-SIG needs to support the library committee in this request and help emphasize the need for such a person to provide technical support for the librarian.

Steven Stepanek and Pat Kleinhammer of California State University at Northridge have volunteered to "clean up" the RSTS portion of the DECUS library and produce a good magtape version of it. Hopefully, they will have good news to report at Spring DECUS.

Any users with comments or suggestions are invited to send them to

Ms. Ardoth Wilson
Computer Center
Central State University
Edmond, Oklahoma 73034

Central State University will have students graduating with B.S. degress in Computer Science (with options including math, accounting, management, etc) in December, May and August. Those desiring to interview prospective graduates should contact:

Ray Beasley, Chairman
Computer Science
Central State University
Edmond, Oklahoma 73034
(405) 341-2980 X761

RSTS EDUCATIONAL SUBSIG

Howard B. Kipple

For those educational RSTS users that were present at the Fall '76 DECUS meeting in Las Vegas, I promise to do everything within my power to get our meeting scheduled prior to the WISH LIST session and not in conflict with the informal RSTS sessions for future DECUS meetings. And for those educational RSTS users that were not present at the Fall '76 DECUS meeting, I want you to know that you missed some very good presentations on some very pertinent topics. I also want you to know that I am aware of restricted travel budgets and that attendance at all DECUS meetings is impossible for some of you, but I do hope that you will attend, and most importantly, participate in the DECUS meetings as often as you can.

During this DECUS meeting the WISE software users met to discuss this software package and I am hoping that a session for these users will be scheduled at future DECUS meetings. All it takes is for someone to submit a paper when the call for papers for the spring meeting is requested. I believe John Nunnally or Mark Grundler will se that this happens.

Another special session that hopefully will appear in the Spring DECUS meeting is a DECAL user discussion. There is sufficient interest in this and other CAI/CMI author languages to warrant a session. I believe that Charles Mustain of Berea City Schools will see that a session is scheduled for the spring meeting. David Hamlin of Tulsa Junior College gave a very fine presentation on how his college is meeting the CAI/CMI needs and would be a good source for people interested in this area.

The EDUSIG business meeting ended with the election of a new chairman and an editor for EDUSIG. The new chairman is Mike Nakoff whose address is Cincinnati Bible Seminary, 2700 Glenwood Ave., Cincinnati, Ohio 45204, and the new EDUSIG editor is Bruce Alcock whose address is Riverdale Country School, W. 253rd St. and Fieldston Rd., Bronx, New York 10471. The people in attendance at the business meeting thought the chairman should decide the remaining officer structure of EDUSIG, so Mike Nakoff may be contacting you to assist him, particularly with the coming spring meeting in Boston. I am sure Mike would appreciate hearing what your specific interests are for the spring DECUS meeting so write and tell him.

There is still a lot of interest from schools for a statistical package. The educational RSTS users really missed hearing from Michael Hall of Lawrence University on the status of COSAP II; however, if you have since read the November 1976 issue of the RSTS newsletter that may explain why Mike was not at the Fall '76 DECUS meeting to talk about COSAP II. Hopefully, something can be worked out and COSAP II released.

For schools that have wanted to use the Statistical Package for Social Studies (SPSS), you may be in luck. Carleton College at Northfield, Minnesota has SPSS running on their PDP II. I talked to Carl Henry at the Fall DECUS meeting about the package. I believe it is written in FORTRAN and run under the FORTRAN run time system under RSTS. If you are interested you should write Carl Henry, Computer Center, Carleton College, Northfield, Minnesota 55057.

LARGE RSTS SYSTEMS SUBSIG

The Large RSTS Systems SubSig had its first meeting in Las Vegas. After much discussion it was decided that the definition of a large RSTS system was any RSTS system that could be considered sophisticated.

The main objectives of the group initially are to effect improvements in system performance via performance measurement, increase the usefulness of CUSPS, and to ease future system limitations.

The major concern was how to measure system performance so that the resulting data could be used effectively to improve that performance. Carl Marbach volunteered as technical editor of the newsletter to coordinate and solicit articles in this and other technical areas. Mike Mayfield volunteered to document JSTATS and DSTATS if he received twenty letters requesting him to do so. I urge you to write him.

CUSPS were another important area of discussion. Most RSTS installations modify some CUSPS. Upon discussion it appears that much duplicate work is done in this area since many RSTS users have similar requirements. In the November newsletter Michael Hall wrote of his problems trying to resubmit modified CUSPS to DECUS. To avoid these problems, it has been suggested that a change file be submitted to DECUS. This change file would contain only the changes to the CUSP and could be APPENDED to the original CUSP at the users' option. Jon Spencer has volunteered to be the CUSP Editor for the newsletter so send your changes to either Jon and/or DECUS.

Although we initially touched on only a few topics, there are many others relevant to large RSTS system users. I welcome your contributions and suggestions for newsletter articles, DECUS topics and speakers, and anything else. Please send them to:

Joan Salkin
(Chairman, Large RSTS Systems Group)
Crown Zellerbach
1 Bush Street
San Francisco, California 94119
415/823-5504

Jon Spencer
(CUSP Editor)
PASCAL Computing
432 S. Barrington Ave.
Los Angeles, Ca. 90049

Carl Marbach
(Technical Editor)
Acme-Hardesty Company
910 Fox Pavilion
Jenkintown, Pa. 19046
215/886-0200

Mike Mayfield
(JSTATS, DSTATS Documentation)
Digital Equipment Corporation
871 South Nash ST.
El Segundo, Ca. 90245
213/640-1830

FROM THE TECHNICAL EDITOR

An understanding of what your computer is doing at the basic level is important in evaluating performance (or lack of it) and deciding what can be done to improve it.

This will be the first in a continuing series to help you understand the physiology of a PDP 11. I am not a hardware expert, lots of people think I am not a software expert, but I am willing to stick my neck out and let you have the ideas as I understand them. As an editor, I will start and keep the ball rolling. I will, when offered, submit other comments and articles to the newsletter. I ask that you send them to me so that I may exercise my editor's perogative and put them in some order.

This first article will deal with input and output. All PDP 11 devices have associated with them a set of registers which may be accessed as if these registers were in the main memory. Each device has associated with it at least two registers: a) a control and status register and b) a data buffer register. Some devices have more than two registers. For example, one of the simplest devices attached to the PDP 11 may be the console terminal. It has four addresses or registers associated with it. They are: 1) printer data, 2) printer status, 3) keyboard data, and 4) keyboard status. You will have by now noticed that the console is really two devices: a reader and a printer. To print a character on the console printer, all that is necessary is that you deposit via a move or similar instruction the seven bit ASCII code for the character you wish to print into the printer's data register and it will be printed. The printer status register will contain a bit that will say done or ready. Since the CPU runs faster than the printer can print, it is necessary after printing a character to check to make sure that the character has been printed before depositing the next character in the data buffer register to be printed. In an analogous fashion, when a character is typed at the keyboard, it can be 'read' by reading the data register of the keyboard. After 'reading,' a bit in the status register is set to indicate that no data is currently available in the data register. This means, of course, that you now have read the data that

was in there. Note that reading a character from the keyboard does not echo it on the terminal printer. This must be done by putting the data that has been read into the printer data register thus echoing the character read.

Notice that there are no true I/O instructions; only the transfer from memory location to memory location is necessary in order to do input or output.

Although this type of input/output programming is simple and straightforward, it would be inefficient in the case of trying to control a multi-terminal system. For instance, a system that had 16 keyboards attached to it would necessitate that the controlling program periodically scan the data or control registers of the keyboard to see if anything had been input. If nothing had been input on that keyboard, it could continue to the next keyboard to see if there was any data there to be input. This technique is known as polling. That is, the controlling program polls or asks each terminal if there is data that it has to transmit. If not, the CPU continues around checking each terminal or device to see if there is any action to be taken. Thus, it becomes a totally CPU-bound job to keep checking your devices to see if there is any input to be done. A second and, as you can see, more efficient class of I/O programming is the so-called interrupt directed input and output. In the control and status register of our console terminal, there is a bit that enables an interrupt. That is, when data becomes available in the data register in the case of input or output, it causes the program to interrupt. What happens when the interrupt occurs from the console terminal? At every CPU cycle the PDP 11, via hardware, is checking to see if an interrupt has occurred. The PDP 11 utilizes a priority vectored interrupt scheme. Let's take them one at a time. The priority of an interrupt directs that if there is, in fact, more than one interrupt occurring at the same time which interrupt will be serviced first. If interrupt processing is in progress and a higher priority interrupt occurs, the lower order interrupt service routine will be interrupted by the higher priority interrupt.

The vectored interrupt scheme allows interrupt processing via hardware to occur directed to particular memory locations. For instance, an interrupt caused from the console terminal keyboard causes the interrupt to trap to location 60 while an interrupt from the console terminal printer causes a like interrupt to location 64. At these locations then, the user, in his program, may specify a particular address where the interrupt service routine is then to begin. This allows hardware to trap the particular location and relieve

software of the job of determining which device interrupted and now what to do about it. Thus, when a character is typed on the console keyboard, if the interrupt enable bit has been set in the control and status register for the keyboard, an interrupt will occur. The interrupt will then trap to location 60 where the user could specify a service routine to handle an input type on the console terminal. In a similar fashion, when a power fail is detected by the system, a hardware interrupt is generated with a vector interrupt to location 24. Location 24 directs the processor to the proper service routine to save registers and necessary memory on the power fail sequence. We have now discovered two important concepts about input/output programming on the PDP 11: those of interrupt, priority interrupt and vector direction of the particular interrupt. These are not unique to the PDP 11 and can be found throughout the computer industry.

You will have noticed that it requires the central processor to be involved with every character transmitted to our console terminal. That is, if we wish to transmit to the console terminal data that is located in ten contiguous memory locations, it is necessary for the CPU to be involved with at least ten instructions, and in real fact, many more. The CPU must deposit or move from one real memory location to the data buffer or data register of the particular device each character separately for the transfer. This is all right for a slow device such as the DEC writer, but can dause problems for devices that go quite a bit more rapidly. When you consider that an LA 36 transmits 30 characters a second while a TU 10 magnetic tape transmits 36,000 characters a second, the overhead involved with transmitting characters through a magnetic tape would greatly exceed that of an LA 36. There is another type of I/O transfer available within the PDP 11 hardware. This is the so-called NPR transfer. The PDP 8 refers to a data break facility and a generalized computer term for this is devices that work on a cycle stealing basis. (Non Processor Request) Devices that operate on a cycle stealing basis steal an occasional cycle from the CPU in order to do their input and output. This means that the devices themselves have the ability to do data transfers from memory to the device or from the device to memory without processor intervention. These "smart" devices need only be told by the processor what location in memory to begin transfers from, how many locations to transfer, where on the peripheral device to begin putting the data and then these devices begin doing it by themselves. When they are finished, they automatically interrupt the CPU. Take the situation of transferring a file from disk to memory. The device, in this case the disk, needs to be told at what memory locations to begin putting the data, how many words (Bytes) to transfer, and where on the disk to begin getting the data.

Once these registers are loaded, the I/O begins to take place in the same fashion as a character was printed on the console terminal when the printer register was loaded with the data to be printed. When the number of words that we specified to be transferred has been transferred, the device automatically interrupts the CPU for more information. This is the so-called stealing of a cycle from the CPU. It would be possible to transfer many thousands of Bytes, or words, of information between peripheral devices only utilizing a very few cycles of the CPU. While this transfer is going on, the CPU is free for other instructions.

In the next article, we will deal with more specific high-speed, large data capacity devices, particularly RPO 2, 3, 4, 5 and 6 disk drives. We will also attempt to talk about the Unibus structure, how data is transferred over the Unibus and the Unibus extensions: the Fastbus and Massbus.

For more information about this article and future articles, I suggest the following reading. In 1968 or 1969, DEC published a book called, Introduction to Programming. Although this book is based on the PDP 8 series of computers, many of the concepts have carried over to the PDP 11. Next, the PDP 11 Peripherals Handbook contains in its introduction and beginning chapters useful information on programming peripheral devices. The addresses and use of different control and status registers for different devices can be found in this particular book. I have a 1973 version of the Peripherals Handbook as well as the 1976 version and they are somewhat different. If you can get both of them, you are that much better off. In addition to the aforementioned, it is also useful to have the Processor Handbook for the particular PDP 11 that you happen to have. It doesn't hurt to have processor handbooks for PDP 11s that you don't have, also. There are differences you will find between the LSI 11 and the PDP 11/70. If you take the time to read and look through some of these books, you will find all of the information that we have talked about here contained in them as well as many examples which space precludes us from putting in the newsletter.

I hope this first article has been interesting to all, useful to some and boring only to a few. I apologize for its inaccuracies and look forward to your comments.

Carl B. Marbach
Acme-Hardesty Co., Inc.
910 Benjamin Fox Pavilion
Jenkintown, Pa. 19046
(215) 886-0200

RSTS/E COMMERCIAL USERS NEWSLETTER
JANUARY, 1977

The attendance at the past DECUS meeting in Las Vegas shows that the makeup of RSTS users, at least at the symposium, is becoming more and more commercially oriented. New commercial sessions this time included Large Systems Workshops, DECNET/E, CTS-500 Product Panel, COBOL V3, Multi-User Business Systems, plus most other sessions had a commercial flair.

Version 6-B of RSTS was of course a major topic of discussion. Many items discussed about a year ago in Dallas were implemented as a direct result of that meeting. A meeting is scheduled in Chicago for late February to provide an educational seminar for 6-B. For more information contact Stuart Katz.

At the Commercial Sub-Sig Meeting, users reaffirmed their desire to continue the Marketplace section of the newsletter. Several changes to the format and purge procedure were recommended for future issues. The revised ground rules are:

(1) Each item will be dated and automatically purged after running for four (4) issues.

(2) Descriptions will be very brief so as to cut down on space required.

(3) No pricing information will be printed.

(4) Software as well as used hardware for sale by DECUS members will be permitted.

In order to start fresh under the new rules, all entries will be purged after this issue. Each member must resubmit for inclusion in future newsletters.

Please send any commercially oriented problems, questions, procedures, techniques, etc., to:

Murray Stinson
Scientific Placement, Inc.
2700 South Post Oak Road, Suite 2430
Houston, Texas 77056

RSTS MARKETPLACE

DESCRIPTION	CONTACT
1.) A virtual sort subroutine for sorting up to 32,000 records for all files (include virtual). 2.) An easy to use line editor for editing program or ASCII data files. (automatic verification) 3.) A management information system for RSTS. 4.) A jobshop scheduler for the allocation of labor and materials in manufacturing or maintenance. 5.) A complete general ledger accounting system includes balance sheet, statements, journals and schedules such as accounts payable. 6.) An interactive payroll package with 941 & W-2. 7.) A complete automotive agency accounting system for GM, FORD and other dealers. An on-line system. 8.) A package of real estate programs - includes amortization schedules, data retrieval and others. 9.) Financial package for bidding and printing holdings of municipal and government bonds.	Al Cordes Houston Timesharing 9039 Katy Freeway #404 Houston, Texas 77024 713-461-5437
1.) Subroutines to phonetically code last names for storage and retrieval via standard phonetic techniques. 2.) Screen generator programs for the Tektronix 4023 buffered CRT terminal. 3.) Complete letter writing system using paragraph/ letter files to build custom upper/lower case letters from form paragraphs. System allows insertions as well as a user composed paragraph.	Murray Stinson Scientific Placement, Inc. 2700 South Post Oak, #2430 Houston, Texas 77056 713-621-9811
Multiple languages can be used with RSTS/E if you have OMSI's RT-11 emulator. OMSI-RT supports MCRO-11, OMSI APL, and other languages, in addition to DEC FORTRAN IV. 64K words of memory are recommended, but there are several 48K users.	Rusty Whitney OMSI Computing 4015 S. W. Canyon Rd. Portland, Oregon 97221 503-248-5903
1.) A consolidating Accounts Payable, General Ledger, Financial Statement package. 2.) A payroll and Labor Distribution package. 3.) An Accounts Receivable and Sales Distribution package.	Henry Stukenborg Plycom Services P. O. Box 160 Plymouth, Indiana 46563 219-936-2183

RSTS Marketplace (cont)

DESCRIPTION	CONTACT
1.) "DATA BOSS" - A completely conversational generalized DBMS with multiple keys. 2.) COUNTRY CLUB PACKAGE - A complete system including accounts receivable, automatic dues, taxes and gratuities, locker control, cashflow analysis, member spend analysis, optional automatic gratuities calculation by waiter, daily managers report, food and beverage analysis, aging, member progress and activity report, all with complete audit trails. 3.) FINSYS - A C.P.A. WRITE UP PACKAGE - Includes Transaction Registers, General Ledger, Multiple Format Profit and Loss Statements, Balance Sheets with comparisons to budget and last year, Employee Earnings Register, 941's, W-2's, Unemployment Reports, Source & Application of Funds. 4.) PAYROLL - Includes hourly and automatic salary inputs, input listings & proofs, up to six miscellaneous deductions per employee. 5.) BILL OF MATERIALS EXPLOSION PACKAGE - With Production Planning and Inventory Control for Manufacturers. 6.) INVENTORY CONTROL SYSTEM - Including Sales Analysis, Reorder points, etc., for wholesalers.	John H. Wright Florida Computer Inc. 99 N. W. 183 Street North Miami, Florida 33169 305-652-1710
1.) GAAP - A General Accounting Application Package. A complete, interactive, on-line, multi-user system supporting the areas of general ledger, accounts receivable, accounts payable, financial reports, and payroll. 2.) DBM - A general purpose Data Base Management Facility. User defined variable length records, variable length fields, and records with repeating groups. 3.) General Utility Package - Includes program for file formatting and listing, checking for possible file corruption, reviewing and modifying the contents of disk blocks, restructuring a data base for increase system efficiency, etc. 4.) An on-line, interactive school administration package including advanced scheduling and grade/attendance reporting capabilities. 5.) A commercial timesharing accounting system - Accumulates raw use data (connect hours, CPU time, storage) by system account number and converts to automatic billing.	Peter Melvin Intelligent System Corp. 460 Tottenpond Rd. Waltham, Massachusetts 02154
GISMO is an index sequential access method for users of DEC's RSTS/E (CTS/E) operating system *Random or sequential add, delete, and update *No searching of overflow chains *Multiple user support *Generic keys	Leonard Farano Gambit Management Strategies 1440 Broadway New York, New York 10018

RSTS Marketplace (cont)

DESCRIPTION	CONTACT
"FIPS" - Financial Information Processing System. An easy to use computerized accounting system designed by accountants for accountants. Includes general ledger, payroll, payables, cost accounting, financial statements, etc.	Michael M. Niernberg Environmental Developers, Inc. 13693 East Iliff Avenue Denver, Colorado 80232 303-755-3377
"PRODUCT 3" - Data Base Management System. Featuring Relative, Keyed Sequential (ISAM), Multi-list chained Sequential access methods, plus multi-user on-line data base management. Available for RSTS/E, RSX11-D, and RSX11-M.	ELS Systems Engineering 2800 Mayfield Road, #203 Cleveland Heights, Ohio 44118
1.) TAM, a format-language-driven terminal-handling front-end software module for transaction-processing application under RSTS/E. 2.) KDSS, a key-to-disk data entry subsystem under RSTS/E. (In both TAM and KDSS, the performance-critical portions are implemented in PDP-11 assembly language and installed under RSTS/E as run-time systems.) 3.) FSORT, a functional equivalent to the SQWIK and MQWIK modules of the RSTS-11 SORT (except that FSORT handles 64K records, rather than 32K), implemented in PDP-11 assembly language as a run-time system under RSTS/E to run anywhere from 8 to 30 times as fast as its BASIC-PLUS counterparts. 4.) LABX, a software package supporting the DEC LPS-11 Laboratory Peripheral System hardware under RSTS/E. 5.) INTCOM, a set of BASIC-PLUS programs allowing two RSTS/E systems to communicate, including ASCII file transfer, via telephone line up to 300 baud.	Tom Evans Evans Griffiths & Hart, Inc. 1844 Massachusetts Ave. Lexington, Massachusetts 02173
1.) A sophisticated flexible and efficient line and text editor. 2.) Maps - financial planning and reporting language.	Ross Systems 1900 Embracadero Rd. Suite 108 Palo Alto, California 94303 415-326-1054
RSTS QED text editor - This QED is an adaptation of the standard QED type editor, written in basic for RSTS, and is a fast, extremely versatile program. Some of the features available in this editor are: *Simple command forms. *Simple access to any line in the file. *Versatile and powerful character-string matching capabilities. *Easy file merging and file splitting.	Marc L. Cohen Four-Site Computer Corp. 39 Rosemont Terrace West Orange, New Jersey 07052 201-736-9078

RSTS Marketplace (cont)

DESCRIPTION	CONTACT
FN1976 - A Sequential Access Method -for multiple (up to 12) TYPE-1 HEADER FILES -input, output, add-on and update functions -any combination of logical record lengths, automatic fielding -package includes MERG76, a generalized 2 to 10-way merge to disc or tape	Dave Mallery Nationwide Data Dialog, Inc. 70 James Way Southampton, Pennsylvania 18966 215-322-2050
The Statistical Programs for the Social Sciences/Carleton (SPSS/C) for the PDP-11. It is written in FORTRAN IV and tested under DOS V9, RT-11, and RSTS.	Graham Kimble or Carl Henry Computer Center Carleton College Northfield, Minnesota 55057 507-645-4431 Ext. 204, 504

BASIC-PLUS STANDARDS

The following is a copy of the BASIC-PLUS Standards document used internally by DEC. It is presented here in the hope that it will serve as a starting point for a user developed standard for BASIC-PLUS. Please send any comments, extensions, or your own ideas on standards to:

Roy Swatzell Jr.
VA Hospital
Room Z R12
Birmingham, Al. 35233
(205) 933-8103 Ext. 239

BASIC-PLUS SOFTWARE CONVENTIONS

Revision 21

07-October-75

G.O. Stone

J. Wooldridge

Revision Notes

I. Changes from revision 1 to revision 2.

A. An example program has been added (Section 5.0).

II. Changes from revision 2 to revision 3.

A. A uniform specification applying to the RSTS/E group and the automated manufacturing systems group was adapted from revision 2. Specific changes are too numerous to mention. The major changes are:

1. New title format for lines 1-299.
2. New variables for standard functions
3. Group specific appendices.

Table of Contents

1.0 INTRODUCTION

2.0 LINE FORMATS

2.1 General Comment Lines

2.2 Block Title Comment Lines

2.3 User Code Lines

3.0 PROGRAM ORGANIZATION AND RESERVED LINE NUMBERS

3.1 Title Block

3.2 Modification History

3.3 Program Description Block

3.3.1 General Description

3.3.2 I/O Channels

3.3.3 Variable Definitions

3.3.4 Function and Subroutine Descriptions

3.4 Dimension Statements

3.5 Program Start

3.5.1 Error Trap

3.5.2 Header Print

3.6 User Code Organization

3.6.1 Constant Definitions

3.6.2 Mainline Code

3.6.3 Program Completion

3.6.4 Programmer Defined Functions and Subroutines

3.6.5 Error Handler

3.7 Standard Routines

3.8 Standard Entry Lines

3.9 End Statement

4.0 USER CODE FORMAT

4.1 Line Format

4.2 Comments

4.2.1 Program Paragraphing

4.2.2 Commenting Rules

5.0 FUNCTIONS AND SUBROUTINES

5.1 User Defined Functions and Subroutines

5.2 Standard Routines

5.2.1 Standard Routine Library

5.2.2 Commenting Standard Routine

6.0 RESERVED VARIABLE NAMES

Appendices

A. References

B. RSTS/E Development Group

B.1 Group Dependent Title Block

B.2 Group Dependent Title Printing

C. Automated Manufacturing Systems Group (Applied Programming)

C.1 Group Dependent Title Block

C.2 Group Dependent Title Printing

C.3 Additional Reserved Variable Names

1.0 INTRODUCTION

This specification defines the software conventions that every BASIC PLUS program written by DEC personnel to be run under RSTS/E must follow. These conventions include coding and program documentation/commenting techniques. The conventions are intended to ease software maintenance, to ease software release activities, to aid the software librarian, and to allow the use of standard routines.

It is important to understand the very real trade-offs between good commenting and efficient coding in BASIC PLUS programs. For example, by using many multi-statement lines, memory requirements are reduced and execution speed is increased. However, it is difficult to thoroughly comment such lines, and it is often impossible to do so without incurring further memory/speed penalites. In many cases, it really comes down to trade-offs between efficient code and understandable code.

All new BASIC PLUS programs must follow the conventions described below. When a program is extensively modified, the new code should follow these conventions.

It should be emphasized that no claim is made that the conventions described in this document are the "best" conventions for providing documentable, maintainable BASIC PLUS programs. Because of the flexibility of BASIC PLUS, a large number of different sets of conventions are equally good. However, this set of conventions reflects a combination of experience and current usage on the part of the authors and others. In the interest of compatibility, this single set of conventions must be chosen and used.

2.0 LINE FORMATS

There are three types of line formats. The first is used for general commenting. The second type is used for comments that are used to delineate a block of code. The third type is used for all user code.

2.1 General Comment Lines

Any line that contains just a comment and which can be deleted from the source without affecting program execution is defined as a general comment line.

General comment lines are in the following format:

NN!NN!(TAB)(TEXT)

Where:

NNNNN is a legal BASIC PLUS line number.

Text is the text of the comment. This text is free form except for the special cases defined elsewhere in this document. Note that all continuation lines must start with a tab.

This format simplifies the automatic extraction of general comment lines from the source code. This allows programs to be compacted before compilation. Compacting may only be done in cases where the program is otherwise too big to compile.

2.2 Block Title Comment Lines

User code is partitioned into blocks or paragraphs (see 4.1). Each such block requires a block title comment line. User defined routines also have a block title comment line (see 5.0). This line must have the following format:

NNNNN(TAB)!(LF)(LF)(LF)(TAB)!(TAB)E X A M P L E T I T L E(LF)(LF)(LF)(CR)

After trying many formats, this exact format was selected. Note that four spaces are needed between words for adequate separation. Note: These lines may not be removed during program compaction (see 2.1).

2.1 User Code Lines

The remaining lines in a BASIC PLUS program are lines that contain BASIC PLUS statements along with comments. These lines must be in the following format:

```
NNNNN(TAB)(CODE LINE)          <OPTIONAL>
   (TAB)(CODE LINE)
          .
          .
          .
(TAB)!(SPACE)(COMMENT TEXT)
(TAB)!(SPACE)(COMMENT TEXT)    <OPTIONAL>
          .
          .
          .
(BLANK LINE)
```

After trying many formats, this format was selected. See section 4 for further discussion of this format.

3.0 PROGRAM ORGANIZATION AND RESERVED LINE NUMBERS

This section describes the overall structure of a BASIC PLUS program. Line numbers assigned to one of the blocks discussed below may be used only for that block. If, for some reason, a block is not needed in a given program, its line numbers may not be used in that program.

BASIC PLUS supplies powerful control structures (some of them currently undocumented). Specifically, it is recommended that use be made of WHILE-NEXT and UNTIL-NEXT loops as well as FOR-NEXT loops, rather than using GOTO statements to control essentially iterative routines.

"It is recommended that programs will be organized on the listing such that they flow down the page, even at the cost of an extra branch of jump." (See appendix A.)

Note that the first line of each reserved block of line numbers is in a standard format (see example). To speed coding programs, these first lines and the copyright statement are provided in an 'APPEND'ABLE utility file.

3.1 Title Block

Lines 1 to 9 are reserved for the title block. Line 1 is a block title comment line (see 2.2) and lines 2 through 9 are general comment lines (see 2.1). The textual content and format of these lines are group dependent (see Appendices).

Lines 11 to 19 are reserved for the standard copyright statement.

3.2 Modification History

Lines 20-99 are to be used to maintain a modification history of the program. Line 20 is the standard block title comment line (see example). The format of these lines will be:

21!<TAB>V/E<TAB>DD-MMM-YYY<TAB>PURPOSE

Where V/E is the new version/edit number, DD-MMM-YY is the date of the modification, and purpose is a brief description of the purpose of the modification. Before the program goes to final release, these lines should be removed.

3.3 Program Description Block

3.3.1 General Description

Lines 100 to 299 are reserved for a general description of the program. The description should be detailed enough so that the reader has a good functional understanding of the program. Unusual formats (such as common core fields, disk record fields, etc. that would be constantly referenced by some one debugging this program) should be given here. The inclusion of formats that are specified in the program specification is optional; the programmer should include those formats that he feels will be most needed by maintenance personnel.

Section 6.0 contains an example of a general description.

3.3.2 I/O Channels

Lines 300 to 399 must contain an ordered list of all I/O channels used by the program. The use of each channel is to be fully explained as shown in the attached sample program.

3.3.3 Variable Definitions

Lines 400 to 799 are reserved to define every BASIC PLUS variable used by the program. These variables must be in alphabetical order. Refer to the attached sample program for suggested ways of defining working variables, etc. Variables which only appear in the standard routines (see section 2.9, 4.0) should not be included in this list. Also, dummy function variables used by any functions should not be included in the list (except when used elsewhere in the program.) Variables used in dimension statements are included in the list and distinguished by placing "()" immediately after the variable name (e.g., A1()). The use of the array should be fully explained either in this section or in the general description section. An attempt to define one variable per line and to evenly distribute the variables over the range of available line numbers should be made in order to speed later additions.

3.3.4 Function and Subroutines Descriptions

Lines 800 to 899 must contain a list of all functions and subroutines used by the program. For each routine (user-defined or standard) used, the title and the starting line number will be included here as a general comment line. User defined routines must be entered from lines 801 to 849.

3.4 Dimension Statements

Lines 900 to 998 are reserved for dimension statements. All dimension statements must be placed here. Each array dimensioned should be briefly described in a comment line. Arrays used by included standard routines will be entered by the 'APPEND' at lines 950-998. The programmer should not use these line numbers.

3.5 Program Start

3.5.1 Error Trap

The program must start at line 1000. The statement at this line should be:

```
1000 ON ERROR GOTO 19000
     ! SET UP STANDARD ERROR TRAP
```

3.5.2 Header Print

Any program which can be invoked directly by the user through a 'RUN' command must use lines 1010-1020 to print the standard header line on the user's terminal. The format of this header line is group dependent (see Appendices).

3.6 User Code Organization

All other user code (except optionally, program completion) must be located in the range from line number 1030 to 19999. It is recommended that lines of code be assigned line numbers in increments of 10; comments need have only an increment of 1.

3.6.1 Constant Definitions

Any constants used by the program should be defined and initialized immediately after line 1030. Because of the logic of BASIC PLUS, it is efficient to use a variable to hold often used constants, rather than using literals. If the programmer wishes to do this, the variables should be initialized here, with appropriate commenting. In some cusps (e.g., SPOOL, LOGIN) certain parameters are set as constants, and the documentation includes a description of how to change this parameter by altering the line. All such constants should be initialized in this first part of the program, and this initialization

should be the only operation on the line, except the comment describing exactly what is being initialized.

All numeric constants should always be terminated with a "," or a "%", as appropriate. This eliminates a source of ambiguity in the text.

3.6.2 Mainline Code

The mainline code includes all non-subroutine and non-function code. It should be paragraphed as described in section 4.

3.6.3 Program Completion

The program completion routines (close of files, self-kill, etc.) will be a separate paragraph of code. It may reside with the main-line code (before line 10000) or it may be located at lines 32700-32760. If it resides with the main code, it is recommended that it start at line 9000. It is highly recommended that a block of line numbers be left unused after the bulk of the main-line code and before the completion code to allow for program expansion.

3.6.4 Programmer Defined Functions and Subroutines

Lines 10000 to 18999 are reserved for programmer defined functions and subroutines. All such functions and subroutines should be located in this reserved space. Subroutines should come first, starting at line 10000; functions follow the subroutines. It is advised that a block of line numbers be left free between subroutines and functions to allow for future expansion.

3.6.5 Error Handler

Lines 19000 to 19999 are reserved for error handling. Separate modules of the program may use their own error handlers, but the trap line number should always be reset to 19000 upon exit. The standard routines will trap their own errors, where appropriate, and reset the error trap to 19000 upon exit.

3.7 Standard Routines

Lines 20000 to 29999 are reserved for use by standard routines (see section 5.2).

3.8 Standard Entry Lines

Lines numbered 30000, 31000, and 32000 have special functions as entry-points to the program from CCL commands, 'CHAINING' to or from CUSP programs, and logged out terminals. In CUSPS, these lines are used as follows:

```
30000 CCL ENTRY
31000 CHAIN ENTRY
32000 LOGGED OUT ENTRY
```

3.9 END Statement

Line 32767 is reserved for use as the "END" statement. No other line is allowed to contain an END statement, and no other statement may be present on line 32767.

4.0 USER CODE FORMAT

4.1 Line Format

The line number of a BASIC PLUS code line will always be followed by a (tab) character. Therefore, lines of BASIC PLUS code will be aligned at the first tab stop, except for some special cases outlined below.

The following suggestions are highly recommended:

1. When a continuation line begins with a new command, put the colon separating it from the previous command on the new line, not on the previous line. The colon is to follow one tab character.

2. Where practical, further indent continuation lines which are part of the previous command, e.g., 'IF modifiers' which run on to a second line.

3. Where practical, the code within a loop should be indented one tab from the statements controlling the loop. (See lines 1600 to 1640 of example program.) Indent the comment to align with the statements.

4. If comments are put on continuation lines or separately numbered lines, indent them by two (tab) characters, so that they line up but are not aligned with the code lines - this makes the comments stand out more clearly (also see 2.1).

Except for 2 and 3 above, it is mandatory that the code be aligned one (tab) from the left margin.

4.2 Comments

The primary purpose of commenting code is to lower the cost of software maintenance and release. Since this is best achieved by having all programmers comment their code in a uniform manner, the following conventions have been established. All comments should appear as complete sentences ending with a period, whenever possible.

4.2.1 Program Paragraphing

The general organization of a program is dictated by the requirements outlined in Section 3. Further clarification of the code should be achieved by "paragraphing", where each paragraph is a section of code that performs one cohesive task. Each such paragraph must be separated from the others by means of a block title comment line (see 2.2).

A general description of the code within the paragraph should follow the title comment if the title comment does not sufficiently explain the task that the coce performs. It is reasonable to have this description on consecutive line numbers.

4.2.2 Commenting Rules

Each line of the actual code must be in the format described in section 4.1. Note that the following techniques are mandatory:

1. Each numbered line of code will have at least one comment line describing the effect of the coae (except in the case described in 3 below). Each such numbered line shall be terminated with a blank physical line (except for 3 below). Each non-blank comment line shall have an exclamation point preceeding the comment text (see example).

2. Computed GOTO and GOSUB statements should be commented such that the reason for each branch is explained.

3. Multi-statement lines are allowed only if each statement on the line contributes to a common goal. In such cases, the goal must be thoroughly described in the usual comments following the paragraph of code (see 1 above). (If the techniques used in the paragraph are non-trivial, the algorithm must be documented in the program specification.) Such "paragraphs" of code may be broken over several line numbers should it become necessary, without comments or extra line-feeds between the lines.

5.0 FUNCTIONS AND SUBROUTINES

Functions and subroutines when referred to as a group are called routines. Lines 10000 to 18999 are reserved for user defined routines. Lines 20000 to 29999 are reserved for standard routines, and must not be used by the programmer. Although there are many reasonable ways to organize BASIC-PLUS code, this separation of all routines is specified to standardize all BASIC-PLUS code within DEC.

5.1 User Defined Functions and Subroutines

It is recommended that user defined subroutines be located between lines 10000 and 14999 and user defined functions between 15000 and 18999. It is mandatory that the subroutines precede the functions. Each routine must have a title line. Subroutines must always be entered by a 'GOSUB' to the title line (note that a subroutine's title line must be a block title comment line (see 2.2)). A description should include data returned by the routine and any 'side effects' of the routine (e.g., non-dummy variables affected, arrays used, etc). Other information (like variable definitions, I/O channels used, etc.) should be included in the first 1000 lines of the program rather than in the routine. However, dummy function parameters should be defined following the description of each function.

The first line number of the routine's code in functions and subroutines. The last line of the code block block must be at least 2 greater than the last line of the descriptor block. All coding restrictions stated in the previous sections apply to code contains only the exit statement (FNEND for functions; RETURN for subroutines); this is the only exit statement in the routine. Any routine which traps its own errors must use its first executable line to set its error trap (note that the DEF statement is not executable and must be the only code on its line). If the routine sets an error trap, the line before the exit line is always

```
ON ERROR GOTO 19000
        ! RESET ERROR TRAP.
```

Nothing else may appear on this line. If this line is present, the routine must always exit through it.

An error handler block contains all error handling for the routine. If no trappable errors are possible in the routine, no error handler block is present. The error block follows the main body of the routine (after the 'RETURN' or 'FNEND'). Note that all trapped errors must eventually exit through the 'ON ERROR GOTO 19000' line.

All errors which the routine may generate must be trapped by the routine. The error flag, E%, will be used to return information concerning the error to the calling program (see 6.0).

Programs must never, after trapping an error from inside a subroutine or function, continue processing without going through the exit line for that routine. Code which does this can cause random errors with an extremely high degree of debug difficulty.

Single-line functions may be used only when the operation performed will not produce an error (under usual circumstances). An error trapped from a single-line function must be considered fatal by the calling program.

5.2 Standard Routines

There is a high degree of similarity in the operations of different programs. For example, some programs accept and decode "command lines"; many programs do directory lookups; some access a standard data base. When similar functions are performed by two or more programs, the maintenance problems are reduced if the same routines are used in all of them. Thus, the existence of 'standard' routines.

5.2.1 Standard Routine Library

The 'standard routine library' is a library of program files stored on a designated account. Each of these files holds code for a given oft-used routine. The code from these files may be included in a program by 'APPEND'ING. In order to avoid line number conflicts, these routines have line numbers in the range 20000-29999, and no program may use these line numbers for any other purpose. All standard routines have unique line numbers so that the 'APPEND' operation can be used.

5.2.2 Commenting Standard Routines

Standard routines will have the same documentation structure as does a program and follow the same rules as user defined routines (see 5.1). Specifically, each routine will have a title block, a modification history, and a program description block (ref. 3.1, 3.2, 3.3). All standard routines which use an (internal) array will include the DIM statement for that array.

Remember that lines 951-998 are reserved for this purpose. If the copyright statement is desired, it shall use lines 11 to 19. The programmer is free to use line numbers of his choice for these items except the first line of the title block must fall on a line number that is a multiple of 50. See section 6.0 for rules on variables permitted in standard routines.

6.0 RESERVED VARIABLE NAMES

Because of the lack of 'local' variables in BASIC-PLUS, any non-dummy variables used in a function or subroutine can affect the whole program. Because not all variable in standard routines can be dummy variables, some variable names are reserved to the standard routines and should not be indiscriminately used in the user part of the code, except as dictated by the standard routines. In addition to these variables, certain other variables have been reserved to hold values which will be needed in (almost) every program. These variables must never be used to hold anything but the contents described below.

All variables of all types (integer, floating, string, or arrays of these) whose names start with 'Y' or 'Z' are reserved for use by standard routines.

All 'X' variables of all types are reserved for use as debugging variables. No other use may be made of these variables. They may only be used in explicitly debugging-type statments in the code. X variables will appear in released programs only in such statements.

The two variables 'E%' and 'E$' are used only to hold information concerning the latest error encountered. If, at any time during a program run, E% is non-zero, it indicates that an error condition exists. When E% is used and is 0, no error condition has yet been detected. The standard routines all conform to this rule, and all user routines must conform as well. The string in E$ can be any error-related text. If additional error information is required, the variable E1% may be used to store it. E1%, however, may be used for other things by the program. An error processing routine must reset E% before returning control to normal processing.

Other variables may be reserved on a group dependent basis.

APPENDIX A

REFERENCES

1. PDP-11 Software Engineering Software Standards.

APPENDIX B

RSTS/E DEVELOPMENT GROUP

B.1 Group Dependent Title Block

Line 1 of every BASIC PLUS program is a block title comment line (see section 2.2). The text of this line is the name of the program. Lines 2 through 10 are general comment lines (see section 2.1) as shown below:

```
2!<TAB><TAB>PROGRAM          :   PROGNM
3!<TAB><TAB>PACKAGE          :   PACKNM/VEN
4!<TAB><TAB>MODULE           :   N OF M
5!<TAB><TAB>VERSION          :   VER
6!<TAB><TAB>EDIT             :   ED
7!<TAB><TAB>EDIT DATE        :   DD/MM/YY
8!
9!<TAB><TAB>AUTHOR           :   NAME<LF><CR>
```

Where: PROGNM is the program name, PACKNM/VEN is the name and version number of the package to which this program belongs; N of M is the sequence number (N) of this program within the package and the number of programs (M) in the package; VER and ED are the version and edit number of this program (see below); and NAME is the name or initials of the programmer.

If the program is not part of a package, lines 3 and 4 do not appear. Line 9 does not appear in the release copy of the program (it is removed during the kit generation process - the in-house master sources will always have it).

The format of the version identifier of a program is determined as outlined in the PDP-11 software engineering software standards document (see 9.0). It is the programmer's responsibility to make certain that the key-letter is changed to 'V' before final packaging. Since an 'EDIT' and a 'PATCH' are identical in BASIC PLUS, we do not use the 'PATCH' field.

The version number of a new program is always identical to the version number of RSTS/E under which it will first be released. The version number of an old program may change in either of the following two cases:

1. The program is re-written from scratch, and uses capabilites of later releases of RSTS/E.

2. The program is upgraded to use features available in a later release of RSTS/E.

In any other modification, for example, bug fixes, extensions which

will work under the original RSTS/E, only the edit level will change. Thus, it is possible to determine, from the version number of the program, the earliest version of the operating system under which the program can successfully execute.

Version numbers are assigned to packages in the same manner as they are assigned to programs. Note that if the version number of any program in a package increases, the version number of the package must increase, since the package will no longer run under the earlier version of RSTS/E.

B.2 Group Dependent Title Printing

Any program which can be invoked directly by the user through a "RUN" command must use lines 1010-1020 to print the standard header line on the user's terminal. The header line shall show the following information:

```
PRGNAM<TAB>V/E<TAB>SYSVEN
```

Where PRGNAM is the name and V/E is the version/edit of the program being run, and SYSVEN is the system identification obtained from the 0th error slot in "ERR.SYS". At line 1010, the variable I$ will be set equal to the text defining the V/E number of the program; e.g.,

```
1010 I$="V06A-01"
          ! SET UP VERSION/EDIT FOR HEADER.
```

Line 1020 will do the final creation and printing of the header. If the program may not be invoked directly by a "RUN" command. Lines 1010-1020 should not be used. Separating the V/E out onto a separate line makes it easier to update this string upon patching the program.

Ordinarily, a program which has been entered through a "CHAIN" or a CCL command will not print its header ine. The entry line combined with line 1020 must contain the code to inhibit the printing in this case.

B.3 Additional Reserved Variable Names

The value of variable E0% will indicate the type of entry (i.e., RUN, CCL, CHAIN, or LOGGED-OUT entry) made to the program. The primary uses of this variable are:

1. To detemine whether or not to print the header line;
2. To determine how to prompt and what editing features to allow in the standard GET command line routine.

Since the GCML routine has not been finalized, all values for E0% are not yet known; however, an E0% value of 0 always indicates a run-type entry. In this case, the header line should be printed and GCML should prompt fully.

B.4 Forbidden Usage

The following 4 features of BASIC PLUS may not be used by the RSTS/E development group:

1. 'E format' numeric constants. All numeric constants must be written out in full; e.g.,

   1.E10 is illegal, but
   10000000000. is legal.

2. 'PRINT USING' and 'MAT' statements

3. Long message SEND/RECEIVE

4. Echo control on KB: handling

In short, no "optional features" of BASIC PLUS may be assumed in any CUSP.

B.5 Implied/Default Values

In general, no "defaults" supplied by BASIC PLUS should be used. This rule is reflected in the requirement that numeric constants must be terminated by "," or "%". The following further restrictions also apply:

1. Semicolons in PRINT statements will always be explicitly entered; e.g.,

   PRINT A "TEST" BS

   Is illegal, but

   PRINT A; "TEST"; BS

   is legal.

2. Never use "&" to mean PRINT (BASIC PLUS II uses & for various things, but not as a synonym for PRINT).

3. Comparisons to 0 or non-zero must always be made explicit. For instance,

... WHILE A% is illegal, but

... WHILE A%<>0% is legal.

(BASIC PLUS II may use a different scheme for testing TRUE/FALSE.)

4. All OPEN statements must contain a filename and extension specification, even if they refer to a non-file-structured device.

E.g.,

OPEN "KB:" AS ... is legal, but

OPEN "KB: PIP.CMD"... is legal.

5. Spaces must be used in the usual places; space removeal to compact lines is not allowed under ay circumstances. (BASIC PLUS II will require spaces.)

B.6 Miscellaneous Requirements

1. In order to provide easier transition to BASIC PLUS II, colon (:) will no longer be used as a statement separator. Instead, the backslash (\) character will be used.

2. Upon CCL, logged-out, or RUN entry, each program should execute the following two statements:

```
ON ERROR GOTO 19000
: PRINT IF POS(0%)<>0%
```

Setting the error trap and returning the KB: to the left margin before any other processing takes place. (Note, in the RUN entry, the PRINT IF POS(0%)<>0% command occurs on the same lne as the header print.)

3. Because of the new CCL mechanism, all programs which have a CCL entry must check for a full length CCL command (starting from the full length) to extract the characters of the command.

4. Because BASIC PLUS II will use the "!" character as a begin and end of comment delimiter, as apposed to simply a begin of comment delimiter as in BASIC PLUS, only one exclamation point may appear in any comment line.

APPENDIX C

AUTOMATED MANUFACTURING SYSTEMS GROUP

(APPLIED PROGRAMMING)

C.1 Group Dependent Title Block

Line 1 of every program is a block title comment line (see 2.20. The text of this line if the program title.

Lines 2 through 9 are general comment lines. Line 2 contains the program file name used by the library and by the application system for storing and running the program.

Line 5 contains the version number.

Line 6 contains the programmer's name.

Line 6 contains the "releasers" name and lines 8 and 9 contain "patchers" names.

The following format must be followed exactly for lines 2 to 91.

```
2¦          FILE NAME:        FILENM.BAS

5¦          REVISION X        DD-MMM-YY

6¦          PROGRAMMER:       JOHN DOE
7¦          RELEASER:         JOHN DOE
8¦          PATCHER:          JOHN DOE
9¦          PATCHER:          JOHN DOE(LF)(CR)
```

Lines 8 and 9 are only included when patches are made to a previously released program.

C.2 Group Dependent Title Printing

The title printed by the code in lines 1010 to 1020 shall conform to the dialogue shown in the system specification for the program.

C.3 Additional Reserved Variable Names

The variables D0% to D9%, E0% to E2%, and d0$ are reserved for use by the standard routine SBINIT. This routine sets D0% to E2% to the values 1 to 12 and D0$ to a null string. All programs should call SBINIT immediately after line 1030 (see 3.6.1) and after all chain entry points. All standard routines can depend upon these variables being set as indicated.

BILL TABOR RSTS SIG
% DECUS OFFICE
146 MAIN STREET
MAYNARD, MASS. 01754

RSTS USER PROFILE

Date of Installation: _____

Contact Person _____

Address: _____

City, State, Zip _____

Company _____ Phone _____

For multiple installations please reporduce this form and respond for each installation.

Respond with quantity of each item.

Processor: 11/20 _____ 11/40 _____ 11/45 _____ 11/50 _____ 11/70 _____ Other _____

Operating System: V4A ____ V5A _____ V5B _____ V5C _____ V6a ____ V6B ____ Other _____

Languages: COBOL _____ Basic + _____ Fortran _____ Dibol _____ Other _____

Memory Size: MOS _____ Cache _____ Core _____ Bipole _____ Other _____
(WORDS)

Vendor of other _____.

RJE 2780 Corrected to: 360 ____ 370 _____ Other _____

Card Reader: CR11 _____ CM _____ Other _____

Card Punch: _____

Line Printer: LS 11 _____ LP 11 _____ Other _____

Terminals: TTY _____ LA 30 _____ LA 36 _____ Other _____

Vendor _____.

CRTS: VT 05 _____ VT 50 _____ VT 52 _____ Other _____ Vendor _____

Lines: DH 11 _____ DN 11 _____ DL 11/A _____ DL 11/E _____ Other _____

Disk Drives: RK 02 _____ RK 05 _____ RP 02 _____ RP 03 _____ RP 04 _____

RS _____ RF _____ Other _____ Vendor _____

Mag Tape: 7 Track _____ TU 10 _____ TU 16 _____ 9 Track _____

Other _____ Vendor _____

DECTAPE: TU 56 _____

Floppy Disk _____

Current Applications: _____

_____

Need Software For: _____

_____

_____

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

DECUS

DIGITAL EQUIPMENT COMPUTER USERS SOCIETY
MAYNARD, MASSACHUSETTS 01754